Conditional Random Fields for XML Applications

Rémi Gilleron, Florent Jousse, Marc Tommasi, Isabelle Tellier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Conditional Random Fields for XML Applications

Rémi Gilleron — Florent Jousse — Marc Tommasi — Isabelle Tellier

N° 6738

November 2008

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6738--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
LILLE - NORD EUROPE

Conditional Random Fields for XML Applications

Rémi Gilleron , Florent Jousse , Marc Tommasi , Isabelle Tellier *

Thème SYM — Systèmes symboliques
Équipe-Projet Mostrare

Rapport de recherche n° 6738 — November 2008 — 36 pages

Abstract: XML tree labeling is the problem of classifying elements in XML documents. It is a fundamental task for applications like XML transformation, schema matching, and information extraction. In this paper we propose XCRFs, conditional random fields for XML tree labeling. Dealing with trees often raises complexity problems. We describe optimization methods by means of constraints and combination techniques that allow XCRFs to be used in real tasks and in interactive machine learning programs. We show that domain knowledge in XML applications easily transfers in XCRFs thanks to constraints and combination of XCRFs. We describe an approach based on XCRF to learn tree transformations. The approach allows to solve XML data integration tasks and restructuration tasks. We have developed an open source toolbox for XCRFs. We use it to propose a Web service for the generation of personalized RSS feeds from HTML pages.

Key-words: Conditional Random Fields, Graphical Models, XML trees, XML Transformations, Web Information Extraction, Data Integration.

* This work was supported by the Marmota project ANR-05-MMSA-0016 and the Atash project ANR-05-RNTL00102

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne
40, avenue Halley, 59650 Villeneuve d'Ascq
Téléphone : +33 3 59 57 78 00 — Télécopie : +33 3 59 57 78 50

Champs Conditionnels Aléatoires pour XML

Résumé : Nous considérons le problème de l'annotation de noeuds dans des arbres XML. Ce problème est fondamental pour de nombreuses applications comme la transformation d'arbres, l'extraction d'informations et l'intégration de données. Nous proposons de résoudre ce problème d'annotation à l'aide de champs conditionnels aléatoires spécialement conçus pour les arbres XML: les XCRFs. Des techniques d'optimisation basées sur des contraintes et des compositions sont ajoutées aux XCRFs pour permettre un passage à l'échelle et un usage dans un cadre interactif. Nous montrons également que ces techniques sont aussi un moyen d'injecter dans le modèle des connaissances du domaine d'application. Nous proposons également une approche originale pour l'apprentissage automatique de transformations d'arbres, basée sur les XCRFs. Nous l'appliquons à des tâches d'integration de données et de restructuration d'arbres XML. L'implantation des XCRFs est diffusée sous licence libre. Elle inclut une interface par service web et sert de base à l'application interactive R2S2 de création de flux RSS personnalisés.

Mots-clés : Champs Conditionnels Aléatoires, Modèles Graphiques, Arbres XML, Transformations XML, Extraction d'informations, Intégration de données.

1 Introduction

The XML format has become the de facto standard for electronic data exchange. XML standard represents documents as a hierarchical structure of elements on top of unstructured data values. This is why XML documents are often called semi-structured and can be formally represented by labeled trees. Node labels are XML elements — or tags, and leafs contain data values.

XML tree labeling is the problem of classifying elements of XML documents, a fundamental task for applications in XML transformation, schema matching, and Web information extraction. Given an input tree $\mathbf{x}$ (the observable), the problem consists in finding an output tree $\mathbf{y}$ (its labeling) of the same shape but with possibly different node labels.

A first instance of XML tree labeling arises in Web information extraction. A fundamental problem here is to select nodes in HTML documents [Gottlob and Koch, 2004, Gottlob et al., 2004]. This can be understood as an XML tree labeling task, where selected nodes get annotated positively and all others negatively. A second application is learning-based schema matching [Doan et al., 2001, Madhavan et al., 2003, Doan and Halevy, 2005]. In this problem, one has to relabel elements of XML documents satisfying a source schema to types of some target schema. A third application is to annotate nodes in XML trees by edit operations (deletion, insertion, etc), in order to define particular tree transformations that may change tree structure [Chidlovskii and Fuselier, 2005, Gallinari et al., 2005]. In this article, we adapt conditional random fields (CRFs), introduced by Lafferty et al. [2001] for XML tree labeling, to our knowledge for the first time.

Conditional random fields (CRFs) are conditional undirected graphical models, *i.e.* families of conditional probability distributions that factorize into local functions (also called potential functions). Which local functions are to be considered is expressed by an underlying undirected graph. See Sutton and McCallum [2006] for a recent survey. CRFs have been successfully applied to sequence labeling tasks in computational linguistics such as part-of-speech tagging, shallow parsing, but also for information extraction tasks [Sha and Pereira, 2003, McCallum and Li, 2003, Pinto et al., 2003, Sarawagi and Cohen, 2004, Sutton et al., 2004] among others.

We adapt CRFs to XML trees which are ordered unranked trees with attributes and define XCRFs. The maximal cliques of the undirected graph associated with an XML tree are: either 3 nodes (a father with two children which are immediate ordered siblings) or 2 nodes (a node and one of its attributes). Local functions for XCRFs consider dependencies between labels according to these cliques. We propose an efficient implementation of XCRFs in an open source toolbox. We show that XCRFs capture all conditional distributions that are induced by probabilistic tree automata over unranked trees. We compare XCRFs with two baseline models: CRFs where maximal cliques contain two nodes (a father and one of its children or a node and one of its attributes); a maximum entropy model where nodes are considered to be independent for labeling. Experimental results show that XCRFs outperform these two baseline models and allow to solve more difficult XML labeling tasks.

With respect to efficiency, an important advantage of XCRFs for XML tree labeling is that computation time depends only linearly on the size of the input documents, both for labeling and training. Apart from this, the upper time

complexity bounds for XCRFs depend on the factor M^3 where M is the cardinality of the label set. As this factor may be time critical, optimization techniques for XCRFs are required. But, we can note that, in many XML applications, XML documents are typed according to schemas. Consequently, we introduce optimization techniques adapted for XML tasks because they turn schemas to profit.

We introduce constraints [Culota et al., 2006] in XCRFs. Constraints restrict admissible assignments for cliques. They allow to reduce the M^3 factor in time complexity. We show that the reduction can be important for XML tree labeling tasks. We experimentally show that constraints allow to reduce computation time for XCRFs for training and for labeling. We also experimentally show that constraints allow to improve performance when there are few examples for learning. This is because incoherent labelings are not considered. This property is very useful for XML applications because it may be expensive to collect labeled XML trees.

We also introduce combination techniques for XCRFs. The base idea is to decompose the set of labels according to domain knowledge, to use XCRFs on these different sets and then to combine the results. For instance, the hierarchical combination can be defined according to a hierarchy of labels defined by a schema. Experimental results show that combination techniques allow to reduce dramatically labeling time and training time. They also show that these techniques preserve performance of XCRFs for XML tree labeling.

Last, we show by experimental evaluation that XCRFs are able to solve realistic XML tree labeling applications. The first application we consider is learning-based schema matching. We provide a new solution by XCRFs that outperform the LSD system of Doan et al. [2001], which is the best previous solution we are aware of. The second application we study is the structure mapping task of the XML mining challenge [Denoyer and Gallinari, 2006]. We reduce this problem to an XML tree labeling task, in which nodes are annotated by appropriate tree edit operations. Learning algorithms induced by XCRFs achieve very good results with precision and recall above 92%. Previous approaches failed to scale to sufficiently large XML training sets. The third application is the generation of RSS feeds from XHTML documents. We show how XCRFs can be used to define programs for generating RSS feeds. We have defined a Web service R2S2 which allows end-users to define automatically customized RSS feeds. This is useful for Web sites where RSS feeds are not made available.

2 XML Tree Labeling

The XML format is the de facto standard for electronic data exchange, especially on the Web. It comes with a number of languages recommended by the W3C that serve for typing, querying, and transformation of XML documents. In this section, we recall selected aspects of XML and argue the relevance of XML tree labeling.

2.1 XML Trees

XML documents are textual representations of hierarchically structured data collections. Parsing turns them into tree structures containing data values. The

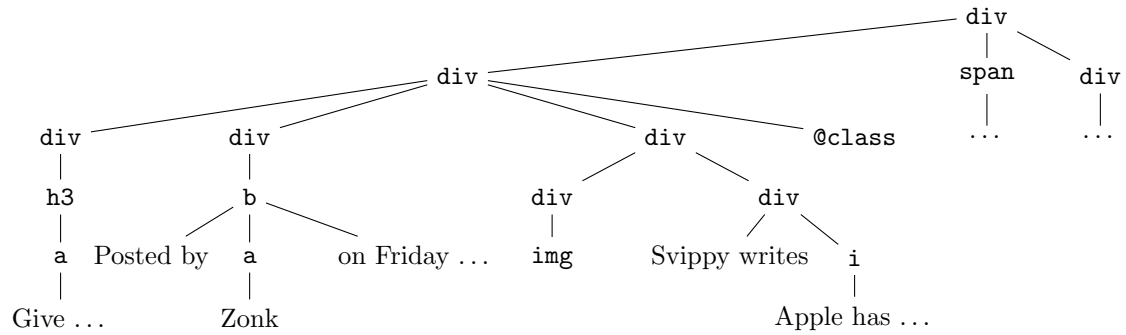

Figure 1: An XHTML tree for a Web document; tags are useful for defining a layout

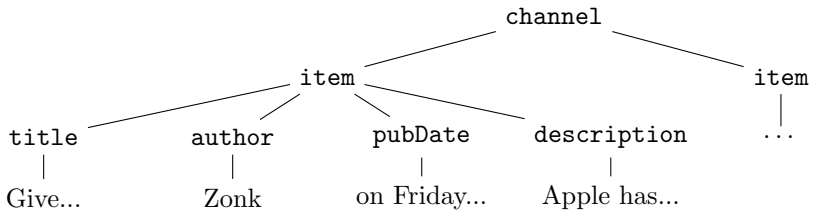

Figure 2: An XML tree according to the RSS 2.0 standard; tags express semantic information

W3C (World Wide Web Consortium) developed a standard representation for XML trees called DOM (Document Object Model). In this representation, XML trees have nodes for elements, attributes, and texts. Element nodes are labeled by XML tags while text nodes contain textual data values. Attribute nodes are labeled by attribute names and contain textual data values. Element and text nodes are sibling ordered, while attribute nodes are unordered. XML trees are unranked in that element nodes may have an unbounded number of children.

XML standard is employed in many applications. In order to constrain the structure of admissible trees in particular applications, XML documents are typed. Typing rules can be expressed by DTDs, XML schemas, and even informally in some specifications. For instance consider the DTD of XHTML for hypertext applications or the specification of RSS 2.0 for RSS feeds. For such applications, node tags express information on the document structure. A first example of an XML tree is the XHTML document in Figure 1. XHTML tags are useful for the layout. There is no relation between an XHTML tag and the semantic of its textual content.

More generally, types can be used to express semantic information about the document contents w.r.t. the target application. A second example of an XML tree is given in Figure 2. Node tags express semantic information about their textual content. Therefore the tree structure is informative. Such XML trees are used in databases applications because the structure allows users to define queries and transformations with languages such as XPATH, XQUERY and XSLT.

We will consider XML applications dealing with these two kinds of XML trees. It should also be noted that XML trees may be large, thus complexity become an important issue.

2.2 XML Tree Labeling

XML tree labeling is the problem of classifying elements of XML documents into a finite set of labels. Formally, given an input tree $\mathbf{x}$ (the observable), the problem consists in finding an output tree $\mathbf{y}$ (its labeling) of the same shape but with new node labels. XML tree labeling has been used for Web information extraction. We now argue the relevance of XML tree labeling for XML applications in the field of XML data integration.

Schema Matching The problem is to find matches between schema elements of a source schema S and schema elements of a target schema T. A simpler problem is to consider 1-1 matches (where a tag in the source schema S has exactly one match in the target schema T). An XML tree labeling program can be used as a matcher. Indeed, if we consider n XML trees $\mathbf{x}_1, \ldots, \mathbf{x}_n$ from the source and their labelings $\mathbf{y}_1, \ldots, \mathbf{y}_n$ from the target, we can define a matcher. Each schema element from S is associated with its most frequent label (a achema element from T). Thus we can define instance level matchers by learning labeling programs from data instances. We will use this technique to compare our method with the schema matching algorithm of Doan [2002] in Section 5.1. It should be noted that XML tree labeling is more general because it allows to label a source schema element into different target schema elements depending on the context of the element in the input tree.

XML Tree Transformation A matcher only defines associations between elements of the two schemas. Thus a second step is necessary to define a program which transforms XML trees according to the source schema into XML trees according to the target schema. This second step is an ad hoc transformation, different for any target schema. Therefore we now show how to learn XML tree transformations in one step. For this, we note that XML tree transformations can be defined from XML tree labelings. The base idea is to consider labels which are tree edit operations. For instance, the mabels can be: relabel the node (*i.e.* give a new tag name according to the target schema), insert a new node with a tag name of the target schema, delete the node or leave the node unchanged. An example of such a labeling is shown in Figure 3. Now, given an input tree and its labeling with tree edit operations, a tree transformation can be defined as shown in Figure 3. Depending on the tree edit operations considered, different classes of XML tree transformations can be defined. This will be discussed in the application section of the paper. But already it could be noted that the more complex is the class of tree transformations, larger is the set of tree edit operations to be considered, and more difficult is learning of the XML tree labeling task.

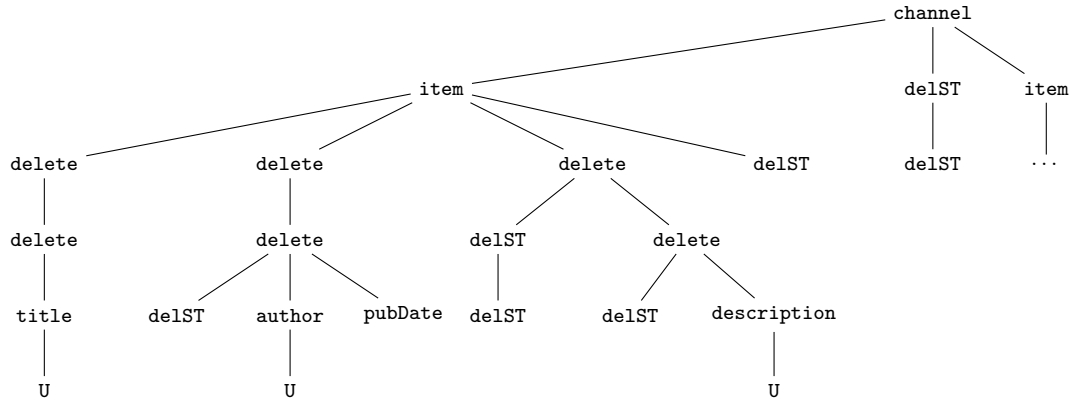

Figure 3: A labeling of the input tree given in Figure 1 where the label `delete` indicates that a node must be deleted, the label `U` says that a node is left unchanged, the label `delST` says that the subtree at this node must be deleted while all other labels are relabeling instructions. Applying these edit operations in a top-down order on the input tree gives the output tree given in Figure 2.

3 Conditional Random Fields for XML Trees

3.1 Introducing XCRFs

We first give the definition of general CRFs before considering the special case of CRFs for XML trees. Let $\mathbf{X} = \{X_1, \ldots, X_N\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_N\}$ be two random fields, *i.e.* two sets of random variables, we denote by $\mathbf{x}$ and $\mathbf{y}$ realizations of $\mathbf{X}$ and $\mathbf{Y}$ respectively. A Conditional Random Field (CRF) Lafferty et al. [2001] is a family of conditional probability distributions that factorize according to an undirected graph $\mathcal{G} = (V, E)$ over $\mathbf{Y}$ (for all $v \in V$, there is a random variable Y_v). If we denote by $\mathcal{C}$ the set of all cliques in $\mathcal{G}$, the conditional probability distribution can be factorized as a product of local functions ψ_c (also called potential functions), $c \in \mathcal{C}$. Each potential function ψ_c takes as parameters $\mathbf{y}_c$, *i.e.* the assignments of the random variables in $\mathbf{Y}$ that belong to the clique c, and the whole observable $\mathbf{x}$. The conditional probability distribution can thus be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c, \mathbf{x}) \tag{3.1}$$

where $Z(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$ is a normalization factor.

The undirected graph encodes the qualitative aspects of the distribution: edges correspond to direct dependencies; two random variables Y_i and Y_j are conditionally independent given some set $\mathbf{Y}_d$, if removing all nodes from $\mathbf{Y}_d$ separates Y_i and Y_j in $\mathcal{G}$. It should be noted that the factorization in Equation 3.1 assumes a node $\mathbf{X}$ and edges $(\mathbf{X}, Y_i)$ in the undirected graph, but they are omitted because they are implicit in CRFs.

The quantitative aspect of the model is defined by the potential functions. It is also assumed in CRFs that potential functions parameterize in a log-linear

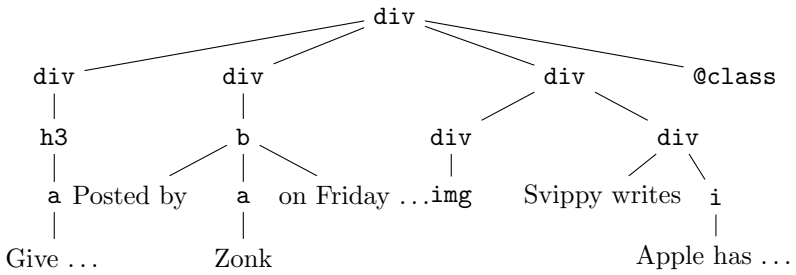

Figure 4: An XML tree $\mathbf{x}$. The set of positions is $\{1, \ldots, 19\}$. $x_1 = x_2 =$`div`; $x_7 =$`b`; $x_{17} =$`i`. In position 19, `@class` is an attribute of the root. The order between the node 19 and its siblings (2, 6, 12) is meaningless while the order between the nodes 2, 6 and 12 is meaningful.

fashion, *i.e.* each potential function has the form:

$$\psi_c(\mathbf{y}_c, \mathbf{x}) = \exp\Big(\sum_k \lambda_k f_k(\mathbf{y}_c, \mathbf{x}, c)\Big)$$

where the model parameters constitute the real-valued vector $\Lambda = \{\lambda_k\}_k$, and $\{f_k\}_k$ is a set of real-valued (often binary-valued) feature functions.

XML documents are ordered unranked trees with attributes. For an XML tree, we identify a node by a position in the set of non negative integers according to a pre-order traversal of the tree. Attributes are unordered and, by convention, we assume that they appear after ordered nodes in the traversal. Thus, for an XML tree, we can define a set of positions $\{1, \ldots, N\}$, we consider the random field $\mathbf{X} = \{X_1, \ldots, X_N\}$ of random variables. For a position n, the symbol x_n is the realization of X_n. An example is given in Figure 4. In the following, we freely identify realizations of such a random field with an XML tree.

We consider tree labeling tasks, the input tree $\mathbf{x}$ and its labeling $\mathbf{y}$ have the same structure and we consider two random fields $\mathbf{X}$ of input random variables and $\mathbf{Y}$ of output random variables indexed by positions of the input tree. We now discuss the choice of CRFs for XML tree labeling tasks, *i.e.* the choice of the undirected graph over random variables in $\mathbf{Y}$. First, in XML trees, there is a vertical recursion which is inherent to the hierarchical nature of trees. Second, there is an horizontal recursion because in XML trees siblings number is unbounded. Types in XML trees implies dependecies following these two recursions. Thus, the undirected graph should generalize over linear-chain graphs for the vertical recursion and it should generalize over linear-chain graphs for the horizontal recursion. Therefore we consider conditional random fields for XML trees (XCRFs) for which the undirected graph is defined by: there are edges between vertices associated with random variables Y_n and Y_i whenever i is a child position of n in the input tree; there are edges between vertices associated with random variables Y_i and Y_j whenever i and j are immediate ordered siblings in the input tree. The undirected graph according to an XCRF for labeling the example tree in Figure 4 is given in Figure 5.

Up to now, we have seen how we define a random field $\mathbf{x}$ from an input XML tree and we have defined the corresponding undirected graph for XCRFs. As for linear chain CRFs, XCRFs extensively rely on parameter tying. Indeed,

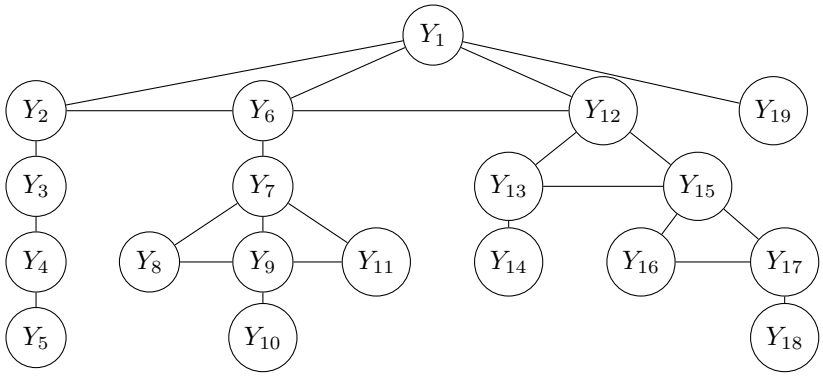

Figure 5: Undirected graph according to an XCRF for labeling the input tree in Fig. 4. Note that the edge between Y_{12} and Y_{19} is not considered because the order between nodes 12 and 19 is meaningless as the node in position 19 corresponds to an attribute.

$$f(y_n, y_i, y_j, \mathbf{x}) = 1 \text{ if } \begin{cases} & y_n = \texttt{delete} \\ \text{and} & y_i = \texttt{delST} \\ \text{and} & y_j = \texttt{description} \\ \text{and} & x_n = \text{div} \end{cases}$$

Figure 6: A feature function for a clique (Y_n, Y_i, Y_j) which can be used in an an XCRF for labeling the input tree in Figure 4. `description` and `delete` are labels of the output tree.

we the same weights are used for feature functions at each position in the tree. This allows to limit the number of parameters and allows to use a given XCRF on every XML tree. As usual, potential functions for XCRFs are defined with feature functions. An example of feature function is given in Figure 6. Each feature function f_k is given a weight λ_k. These weights $\Lambda = \{\lambda_k\}$ are the parameters of the model.

Other CRF models for XML trees could be defined. Considering a more complex undirected graph would lead to a larger class of conditional probability distributions. But exact inference algorithms would become intractable. Also simpler undirected graphs could be considered. For instance we will compare XCRFs with two baseline models: CRFs in which the maximal cliques are pairs (Y_n, Y_i) where Y_i is a child of Y_n; and CRFs or Markov entropy models in which maximal cliques are reduced to single nodes.

3.2 Algorithms for XCRFs

XCRFs are used for labeling XML trees, *i.e.* for computing the most likely (or Viterbi) labeling $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$, given a parameter vector Λ and an input tree $\mathbf{x}$. The parameter vector is computed via parameter estimation: given a sample set $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{i=m}$ of m pairs, learn the vector Λ that best fits S according to some criteria, typically maximum likelihood. This is done by

maximizing conditional log likelihood; several algorithms have been compared for this purpose in Wallach [2002], all based on gradient techniques. These gradient-based techniques require to compute the marginal distributions $p(\mathbf{y}_c|\mathbf{x})$ for all the cliques in the graph together with the normalization factor $Z(\mathbf{x})$. They are computed by inference algorithms and we should note that complexity of learning is strongly dependent on the complexity of inference algorithms.

For general undirected graphical models with cycles, approximate inference algorithms have been defined with sampling based methods or variational methods Jordan et al. [1999], Wainwright and Jordan [2003]. But, the undirected graph for XCRFs is triangulated, therefore, we can consider exact inference algorithms. In the sequel of the section, we propose inference algorithms for XCRFs which are instances of the standard variable elimination/junction tree algorithm for graphical models.

3.2.1 Inference algorithms for XCRFs

The first step is to construct the junction tree associated with the undirected graph. The nodes of the junction tree are labeled by the maximal cliques of the undirected graph. We construct the junction tree according to the preorder traversal of the undirected graph because it corresponds to the internal representation of XML trees. An example is shown in Figure 7. Moreover, in a junction tree, with each node labeled by c corresponds a list of potential functions of some sub-cliques of c. The junction tree must satisfy the property that each potential function is associated with exactly one node. We associate each clique potential with the bottom most node in the junction tree. For instance, as shown in Figure 7, the potential of clique $\{1, 6\}$ is associated with node labeled by $\{1, 6, 9\}$ instead of node labeled by $\{1, 2, 6\}$ since it is the bottom most node. Then, a function $\phi_c(\mathbf{y}_c, \mathbf{x})$ is associated with each node c in the junction tree. This function is the product of the potential functions associated with c. For instance, on our example, for $c = \{1, 6, 9\}$, we have: $\phi_c(\mathbf{y}_c, \mathbf{x}) = \psi_{\{1\}}(\mathbf{y}_{\{1\}}, \mathbf{x}) \times \psi_{\{1,6\}}(\mathbf{y}_{\{1,6\}}, \mathbf{x}) \times \psi_{\{1,9\}}(\mathbf{y}_{\{1,9\}}, \mathbf{x}) \times \psi_{\{1,6,9\}}(\mathbf{y}_{\{1,6,9\}}, \mathbf{x})$.

Once the junction tree is built, the message-passing algorithm can be applied. The goal of the message-passing algorithm is to compute the marginal probabilities of all the maximal cliques (*i.e.* nodes of the junction tree) in the graph, for all their possible label assignments. Marginals are recursively computed using messages exchanged (and memoized) between all the nodes in the junction tree. The message sent from node c' to node c in the junction tree (c and c' being two adjacent cliques in the undirected graph), denoted by $\mu_{c'c}(\mathbf{y}_{c\cap c'}, \mathbf{x})$, takes as input the labels assigned to the nodes which are in both cliques c and c', and the observable $\mathbf{x}$, and is defined by:

$$\mu_{c'c}(\mathbf{y}_{c\cap c'}, \mathbf{x}) = \sum_{\mathbf{y}_{c'\setminus c} \in \mathcal{Y}^{\|c'\setminus c\|}} \phi_{c'}(\mathbf{y}_{c'}, \mathbf{x}) \prod_{c'' \in \mathcal{N}(c')\setminus c} \mu_{c''c'}(\mathbf{y}_{c''\cap c'}, \mathbf{x}) \qquad (3.2)$$

where $\mathcal{N}(c')$ is the set of neighbours of the clique c' in the junction tree. The principle is to sum over all the admissible label assignments for the nodes of c' which are not in c (denoted by $\mathcal{Y}^{\|c'\setminus c\|}$), to compute the potentials in c' and to recursively compute all the messages coming from the neighbours c'' of c'. The algorithm terminates because the graph of neighbours is a tree.

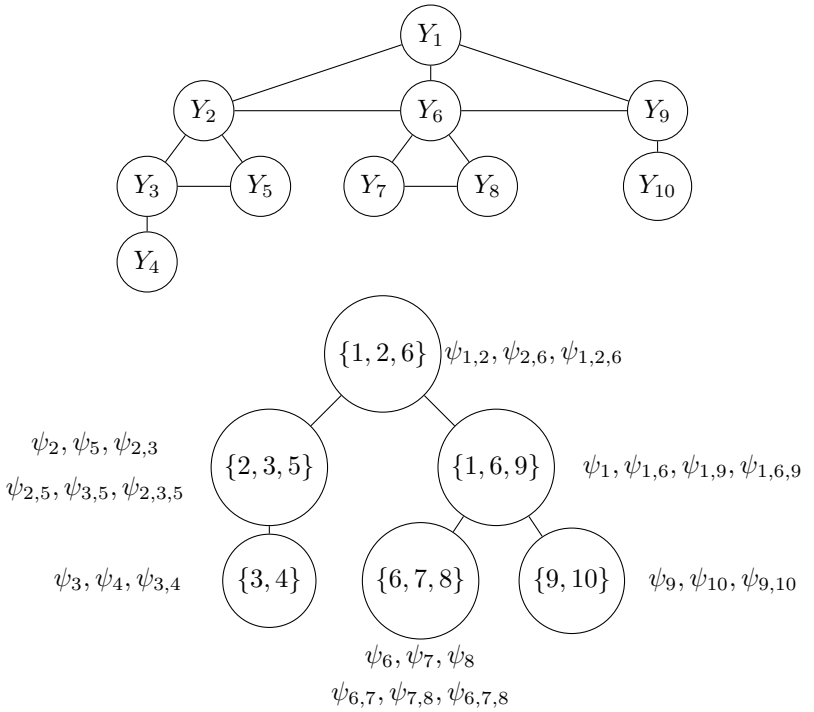

Figure 7: An undirected graph and its junction tree. We simplified notations: $\psi_{9,10}$ stands for $\psi_{\{9,10\}}$

.

The normalization factor $Z(\mathbf{x})$ is a sum over all labelings. It can be written as:

$$Z(\mathbf{x}) = \sum_{y_r \in \mathcal{Z}_r} p(\mathbf{y}_r|\mathbf{x}) = \sum_{y_r \in \mathcal{Z}_r} \phi_r(\mathbf{y}_r, \mathbf{x}) \prod_{c \in \mathcal{N}(r)} \mu_{cr}(\mathbf{y}_{c \cap r}, \mathbf{x}) \tag{3.3}$$

where r is the clique at the root of the junction tree. Therefore, it can be efficiently computed with the message-passing algorithm. The marginal probability of a clique c can be computed as follows:

$$p(\mathbf{y}_c|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \phi_c(\mathbf{y}_c, \mathbf{x}) \prod_{c' \in \mathcal{N}(c)} \mu_{c'c}(\mathbf{y}_{c' \cap c}, \mathbf{x}). \tag{3.4}$$

For decoding, *i.e.* computing the most likely labeling, the Viterbi recursion is obtained by replacing the `sum` function by the `max` function in the message definition in Equation (3.2). Finding the most likely labeling $\hat{\mathbf{y}}$ then consists in the memorization of the Viterbi path associated with the maximum unnormalized conditional probability.

The time complexity is $O(F \times N \times M^3)$, where F is the number of features, N is the size of the junction tree and M is the total number of labels in $\mathcal{Y}$. The factor F comes from the computation of a single potential, while the recursion over all the nodes in the junction tree gives the factor N. Finally, for each node in the junction tree, the algorithm has to sum over all the possible labelings.

Thus the factor M^3 because a node corresponds to a maximal clique of size at most 3. It should be noted that the time complexity is linear in the size of the junction tree. Therefore it is also linear in the size of the input XML tree. This is very important for XML applications because XML trees can be large. However, the factor M^3 may cause problems in the case of large sets of labels. We will propose optimization techniques for XCRFs in Section 4 to deal with this issue.

3.2.2 Parameter estimation for XCRFs

We are given i.i.d. training data S of pairs of the form (observable tree, labeled tree). Parameter estimation is typically performed by maximum likelihood. The conditional log-likelihood, defined as $\mathcal{L}_\Lambda = \sum_{(\mathbf{x},\mathbf{y})\in S} \log p(\mathbf{y}|\mathbf{x};\Lambda)$, is used. This function is concave and the global optimum is the vector of parameters for which the first derivative is null. However, finding analytically the optimimum with respect to all the model parameters is impossible. A gradient ascent (L-BFGS), which requires the calculation of the partial derivatives of $\mathcal{L}_\Lambda$ for each parameter, is therefore used. Replacing $p(\mathbf{y}|\mathbf{x};\Lambda)$ by its definition, $\mathcal{L}_\Lambda$ becomes:

$$\mathcal{L}_\Lambda = \sum_{(\mathbf{x},\mathbf{y})\in S} \sum_{c\in\mathcal{C}} \sum_{k} \lambda_k f_k(\mathbf{y}_c, \mathbf{x}, c) - \sum_{\mathbf{x}\in S} \log Z(\mathbf{x}) \ .$$

Thus partial derivatives can be written as:

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_k} = \sum_{(\mathbf{x},\mathbf{y})\in S} \sum_{c\in\mathcal{C}} f_k(\mathbf{y}_c, \mathbf{x}, c) - \sum_{\mathbf{x}\in S} \sum_{c\in\mathcal{C}} \sum_{\mathbf{y}_c\in\mathcal{Z}_c} p(\mathbf{y}_c|\mathbf{x}) f_k(\mathbf{y}_c, \mathbf{x}, c)$$

The computation of the first term is relatively straightforward. Calculating the second one requires to compute the marginal probabilities for which an algorithm has been presented.

3.2.3 A Library for CRFs for XML Tree Labeling

The library is freely available open source[1].

It allows to define XCRFs in XML. Features can be manually defined via an XML file according to a given XML schema. It should be noted that features must be defined via XPATH expressions. Also the library contains a procedure for the automatic generation of features from a sample of pairs of trees. The procedure is sketched in Section 3.4.

The library includes efficient implementations of inference algorithms and parameter estimation algorithm with penalized maximum likelihood.

The library has been used to define the R2S2 Web service[2] for automatic generation of RSS feeds which is described in Section 5.3. It has also been used in an automatic wrapper induction system from hidden-Web sources described in Senellart et al. [2008].

3.3 Expressivity of XCRFs: XCRFs and Probabilistic Tree Automata

We now discuss the expressiveness of XCRFs by comparing them with probabilistic tree automata. We show that conditional probability distributions defined

[1] `http://treecrf.gforge.inria.fr`
[2] `http://r2s2.lille.inria.fr`

by probabilistic automata can be defined by XCRFs. The result extends a similar result for strings stating that conditional probability distributions defined by hidden Markov models (HMMs, equivalently probabilistic (string) automata) can be defined by linear chain CRFs in McCallum et al. [2000].

As probabilistic tree automata have been defined over sibling-ordered trees, in this section, we ignore attributes and textual contents. Thus XML trees become unranked sibling-ordered trees over a finite alphabet. In the non probabilistic case, different automata models for unranked trees have been defined and compared. Roughly speaking, it has been shown that tree automata for unranked trees can be defined as tree automata over binary encodings of unranked trees. The interested reader can read the chapter 8 of Comon et al. [2007]. The proofs can be extended to the probabilistic case and probability distributions defined by probabilistic automata over unranked trees correspond to probability distributions over their binary encodings. Thus we now consider binary ordered trees in order to compare XCRFs with probabilistic tree automata.

A *weighted tree automaton* Berstel and Reutenauer [1982], Esik and Kuich [2003] over a binary alphabet $\mathcal{X}$ is is a tuple $\mathcal{A} = (Q, I, \Delta)$ where Q is a finite set of states, $I : Q \rightarrow [0, 1]$ is a function assigning initial probabilities to all states, and Δ is a set of transition rules of the following type:

$$r : \ q \rightarrow b(q_1, q_2) \ \ (w) \tag{3.5}$$

where b is a binary symbol in $\mathcal{X}$, $q, q_1, q_2 \in Q$ and $w \in (0, 1]$, or

$$r : \ q \rightarrow a \ \ (w) \tag{3.6}$$

where a is a constant symbol in $\mathcal{X}$, $q \in Q$ and $w \in (0, 1]$.

A *probabilistic tree automaton (*PTA*)* is a weighted tree automaton with local normalization conditions:

$$\sum_{q \in Q} I(q) = 1, \text{ and } \forall q \in Q \sum_{\{r \in \Delta | l(r) = q\}} w(r) = 1 \tag{3.7}$$

where $l(r)$ denotes the left-hand side of rule r and $w(r)$ denotes the weight w of rule r. A run $\mathbf{y}$ over $\mathbf{x}$ is a tree over Q with the same shape than $\mathbf{x}$ that conforms to the rules in Δ. The joint probability $p(\mathbf{x}, \mathbf{y})$ of a tree $\mathbf{x}$ and a run $\mathbf{y}$ is given by the product of the weights of all rules used to define $\mathbf{y}$ times $I(q)$ where q is the root symbol of $\mathbf{y}$. The probability $p(\mathbf{x})$ is the sum over all runs of $\mathcal{A}$ over $\mathbf{x}$[3]. The conditional probability $p(\mathbf{y}|\mathbf{x})$ is defined by $p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})}$.

Now, let us consider a PTA $\mathcal{A} = (Q, F, \Delta)$. We consider the set Q as a set of labels and a run $\mathbf{y}$ of $\mathcal{A}$ on a tree $\mathbf{x}$ can ve viewed as a labeling of $\mathbf{x}$. The PTA $\mathcal{A}$ defines a conditional probability distribution $p(\mathbf{y}|\mathbf{x})$. We show that there exists an XCRF defining the same conditional probability distribution. First, we define XCRF($\mathcal{A}$). For every rule $r : \ q \rightarrow b(q_1, q_2)\ (w) \in \Delta$, we define a Boolean feature function:

$$f_r(y_n, y_i, y_j, \mathbf{x}, c_n) = \begin{cases} 1 & \text{if } y_n = q, y_i = q_1, y_j = q_2, x_n = b \\ 0 & \text{otherwise} \end{cases}$$

[3]Note that there exist PTA such that $\sum_t p(t)$ is strictly lower than 1 (it is a consequence of a similar result for probabilistic context-free grammars given in Wetherell [1980]). Here, we consider PTA defining probability distributions.

where $c_n = (n, i, j)$ is the triangular clique rooted in position n. We set $\lambda_r = \log w$. We complete with feature functions for all b, q_1, q_2, q for which there is no rule in Δ, and set their weights to $-\infty$. We do the same trick for rules of the form $q \to a\ (w)$. Moreover, for every state $q \in Q$, we define a 0-1 valued feature function: $f_q(y_n, \mathbf{x}, c'_n) = 1$ if $y_n = q$ and n is the root, and 0 otherwise, where c'_n is the one-node clique in position n. We set $\lambda_q = \log I(q)$ if $I(q) \neq 0$, and $\lambda_q = -\infty$ otherwise.

We now show that $p_{\text{XCRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x}) = \frac{p_{\mathcal{A}}(\mathbf{y},\mathbf{x})}{\sum_{\mathbf{y}} p_{\mathcal{A}}(\mathbf{y},\mathbf{x})} = p_{\mathcal{A}}(\mathbf{y}|\mathbf{x})$. Let us consider a tree $\mathbf{x}$ and a run $\mathbf{y}$ of $\mathcal{A}$ over $\mathbf{x}$. By definition of $p_{\text{XCRF}(\mathcal{A})}$, we get:

$$p_{\text{XCRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathcal{C}} \prod_{k} \exp(\lambda_k f_k(\mathbf{y}_c, \mathbf{x}, c))$$

For every position n, let us consider the maximal clique c_n rooted in position n. The clique c_n is a single node if n corresponds to a leaf or a triangle clique if n corresponds to an internal node. By construction of $\text{XCRF}(\mathcal{A})$, there is only one function feature which takes value 1: it is the function feature associated with the rule r applied in the run $\mathbf{y}$ over $\mathbf{x}$ in position n. For the root, only the feature function f_{y_ϵ}, where ϵ is the root, takes value 1. Therefore, we get: $p_{\text{CRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} p_{\mathcal{A}}(\mathbf{x}, \mathbf{y})$.

Now, let us consider a tree $\mathbf{y}$ over $\texttt{nodes}(\mathbf{x})$ with labels in Q which do not correspond to a run of $\mathcal{A}$ over $\mathbf{x}$. There is a position p such that no rule applies, thus, by construction of $\text{XCRF}(\mathcal{A})$, there is a feature function with weight $-\infty$ which takes value 1. This leads to a null value for $p_{\text{XCRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x})$. Thus, we get: $Z(\mathbf{x}) = \sum_{\mathbf{y} \in \texttt{runs}(\mathbf{x})} p_{\mathcal{A}}(\mathbf{x}, \mathbf{y})$, leading to $p_{\text{XCRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x}) = p_{\mathcal{A}}(\mathbf{y}|\mathbf{x})$. Thus,

Proposition 1 *A conditional probability distribution over binary trees defined by a* PTA *can be defined by an* XCRF.

As a consequence, it also holds that a conditional probability distribution over ordered unranked trees defined by a PTA can be defined by an XCRF.

3.4 Expressivity of XCRFs: Experimental Evaluation

We now discuss the expressiveness of XCRFs by experimental evaluation. We compare XCRFs with two baseline models: CRFs for trees in which maximal cliques are pairs $\{n, i\}$ where i is a child of n and Markov entropy models in which maximal cliques are reduced to single nodes.

Feature Generation For the comparison to be fair we define a feature generation procedure. Because features must describe properties of input trees, a preprocessing procedure add new attributes to input trees (existing attributes are preserved). These are structure attributes given in Table 1 for internal nodes and text attributes given in Table 2 for text nodes.

Now, we define tests on the input tree in position i of the form

- $x_i = a$, where $a \in \mathcal{X}$, or
- a structure attribute of x_i has a given value, or
- a text attribute of x_i has a given value.

Attribute	Description
nbChildren	number of children of the node
depth	depth of the node
childPos	node is the i^{th} child of its father

Table 1: Structure Attributes (for internal nodes)

Attribute	Description
containsComma	text contains a comma
containsColon	text contains a colon
containsSemiColon	text contains a semi-colon
containsAmpersand	text contains an ampersand
containsArobas	text contains an arobas
isUpperCase	text is in upper case
firstUpperCase	first letter is in upper case
onlyDigits	text is made of digits
oneDigit	text is a single digit
containsDigits	text contains digits
rowHeader	text value of the row header (XHTML only)
columnHeader	text value of the column header (XHTML only)

Table 2: Text Attributes (for text nodes)

For instance, let us consider XML trees in Figure 2 and Figure 3, examples of tests are: $x_i = \texttt{year}$, or attribute nbChildren of x_i has value 2.

For a node n, features contain tests on the input tree for nodes in the neighborhood of n. The neighborhood is controlled by a *neighborhood parameter* j. The neighborhood of a node n is defined as the set of ancestors and siblings of n which distance from n is less than j. The neighborhood parameter measures the quantity of information the XCRF has about the input tree. It will vary in the experiments. It should be noted that we do not consider descendants in the neighborhood because XML trees are unranked (the number of tests would be unbounded).

Let us suppose that the neighborhood parameter is fixed. A *one-node feature* in position n contains a test on the input tree in the neighborhood of n and a test on the label $y_n = l$ where $l \in \mathcal{Y}$. An *edge-node feature* in positions n, i (where i is a child of n) contains a test on the input tree in the neighborhood of n and a test of the form $y_n = l_1 \wedge y_i = l_2$. A *triangle feature* in positions n, i, j (where i, j are immediate ordered siblings and are children of n) contains a test on the input tree in the neighborhood of n and a test of the form $y_n = l_1 \wedge y_i = l_2 \wedge y_j = l_3$.

Given a sample set $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{i=m}$ of m pairs where $\mathbf{y}^{(i)}$ is a labeling of an input XML tree $\mathbf{x}^{(i)}$. We will let vary the neighborhood parameter j. The generation procedure extract all features that appear in the sample for defining an initial XCRF. We will compare XCRFs with baseline models: for 2-CRFs, we only consider one-node features and edge features; for 1-CRFs, we only consider one-node features.

#-examples	Neighborhood	1-CRFs	2-CRFs	XCRFs
5	0	42.92	69.46	**90**
	1	80.93	85.98	**88.07**
	2	79.35	**86.65**	84.76
	3	79.80	**89.51**	87.06
10	0	72.12	86.72	**91.87**
	1	88.29	91.08	**91.55**
	2	85.66	88.60	**94.51**
	3	88.25	88.20	**92.86**
20	0	45.50	78.77	**97.92**
	1	85.18	92.3	**97.17**
	2	90.43	95.88	**97.72**
	3	91.72	94.71	**99.97**
50	0	54.48	89.02	**98.56**
	1	92.95	96.25	**98.73**
	2	93.17	96.83	**99.26**
	3	93.39	96.89	**99.95**

Table 3: Experimental results for label rediscovery on "Courses".

Label Rediscovery Problems We use the "Courses" dataset[4] collected by Doan. It consists of 960 XML documents of about 20 nodes each, containing course information from five universities. In each XML input document, all XML tags (node symbols) are removed and replaced with a unique and therefore uninformative one. Leaf nodes containing text are not altered. The task consists in relabeling the XML documents with the original 14 tags. This task is very easily configurable and reproducible. We let vary the sample size from 5 example pairs to 50 example pairs. We let vary the neighborhood parameter from 0 to 3. We compare XCRFs with 2-CRFs and 1-CRFs. We report the macro-average F_1-measure over all labels which is evaluated on trees not used for learning. Results are presented in Table 3.

The experimental results show that XCRFs outperform the two baseline models. It could also be noted that the influence of neighborhood parameter is less important for XCRFs than for the two baseline models. Also, with 20 examples (2% of the corpus size) XCRFs achieve near perfect results showing the interest of conditional models for the task. The average training time for XCRFs with 20 examples is approximately 15 seconds while it is 3 seconds for 2-CRFs. The average time for annotation for XCRFs is 0.08 second while it is 0.05 second for 2-CRFs.

It should be noted that the task is not so easy because the DTD rule for `course` is:

```
<!ELEMENT course
        ((misc|title|credits|days|time|place|instructor)+,
                                        (section|session)*)>
```

This DTD rule is not very informative because tags under `course` can appear in any order. Nevertheless XCRFs allow to find the correct labels because

[4] `http://anhai.cs.uiuc.edu/archive`

features can express relations between labels. We also do experiments on the label rediscovery task for the corpus Movie which is presented in the next section. Experimental results confirm that XCRFs outperform baseline models, the gain being lower because the problem is simpler (the Movie schema being more constrained).

This first set of experiments shows that the expressivity of XCRFs allows to solve more complex XML tasks. It also shows that XCRFs outperform baseline models. We retain the XCRF model for XML tree labeling tasks. We set the neighborhood parameter to 3 in all further experiments.

4 Optimizing XCRFs

We have proposed XCRFs for which exact inference algorithms have been defined. The good news is that complexity of labeling XML trees is linear in the size of input XML trees. This is important because XML trees may be large. The issue is the factor M^3, where M is the number of labels, in the complexity of labeling XML trees. Indeed, we will see that real XML tree labeling tasks may have to consider large sets of labels. But, for most XML applications, schemas constrain the structure of admissible trees, i.e. domain knowledge is available. In the sequel of the section, we show, in the case of large sets of labels, how to use domain knowledge for optimizing XCRFs. We present how we can add constraints in XCRFs and how we can combine XCRFs.

4.1 XCRFs with constraints

For an XML labeling task and an XML input tree, not all labelings are admissible. For instance, let us consider the labeling task defining an XHTML to RSS transformation illustrated in Figures 1, 2, 3. According to the RSS 2.0 specification, the tag `item` can not appear below the tag `title`. Therefore, if a node is labeled by `title`, its sons can not be labeled by `item`. We introduce constraints that will allow XCRFs to consider only admissible labelings.

A *constraint* is defined by a (possibly empty) test on an observation $\mathbf{x}$ and a set of forbidden labelings. For instance, $(Fl = \{(\texttt{title}, \texttt{item})\})$ is a constraint. It expresses that edge cliques can not be labeled by (`title`,`item`). Another example is $(x_n \text{ is a leaf } \wedge Fl = \mathcal{Y} \setminus \{\perp, \texttt{delST}, \texttt{pubDate}\})$. It expresses that leaves can only be labeled by $\perp$ or `delST` or `pubDate`. We now consider XCRF*s with constraints* as XCRFs together with a set of constraints.

Constraints restrict the sets of admissible labelings for cliques. Constraints satisfy independence conditions defined by XCRFs. Thus, it is easy to modify the message-passing algorithm for constrained XCRFs: only admissible labelings are considered in the messages. Also parameter estimation for constrained XCRFs can be easily defined.

It should be noted that not all labeling properties can be defined by constraints. For instance, let us consider the labeling property: "the label a occurs at most once in the labeling tree". It is easy to show that it can not be defined with constraints. It should also be noted that such a labeling property do not satisfy independence conditions for XCRFs and can not yet be implemented efficiently in XCRFs.

#-examples	Neighborhood	XCRFs	XCRFs with constraints
5	0	90	91.89
	1	88.07	91.88
	2	84.76	90.92
	3	87.06	88.85
10	0	91.87	92.58
	1	91.55	96.58
	2	94.51	96.72
	3	92.86	96.69

Table 4: Comparing XCRFs with and without constraints on the label rediscovery on the Courses dataset.

The interest of XCRFs with constraints is twofold. First, the time complexity of inference is now $O(F \times N \times A)$ where A is the size of the largest set of admissible labelings for triangular cliques. The factor A can be very lower than M^3 when constraints restrict dramatically the set of admissible labelings. But it remains equal to M^3 when there is no constraint. Second, constrained XCRFs allow to eliminate incorrect labelings w.r.t. domain knowledge. Thus XCRFs with constraints can be more accurate than XCRFs. We now present experimental results in order to discuss these two issues.

First, we consider the label rediscovery problem on the Courses dataset. We use the output DTD and define constraints accordingly. For instance, the DTD rule `<!ELEMENT time (starttime,endtime)>` allows to define the constraint $(Fl = \{(\texttt{time}, y_1, y_2) \mid (y_1, y_2) \neq (\texttt{starttime}, \texttt{endtime})\})$. Introducing all constraints according to the output DTD, the size of the largest set of admissible labelings for triangular cliques is now $A = 140$ to be compared with $M^3 = 14^3 = 2744$.

We compare XCRFs with XCRFs with constraints. Experimental results are presented in Table 4. They show that XCRFs with constraints outperform XCRFs when the number of examples for learning is small. This property is interesting in XML applications because it is often expensive to collect examples. We do not present experimental results for larger number of examples because there is no significant difference between XCRFs with or without constraints. The average training time for XCRFs with constraints is 10 per cent lower than for XCRFs on this dataset. This is true also for the average labeling time. It is less than expected when comparing A and M^3. It is because the message-passing algorithm has to do additional tests in order to only consider admissible labelings.

We do other experiments on label rediscovery tasks. They confirm that XCRFs with constraints outperform XCRFs for small datasets. When the output DTD is very informative, the factor A can become very small. For instance on the Movie dataset, we obtain $A = 204$ while $M^3 = 37^3 = 50563$. In this case, the average training time for XCRFs with constraints is 25 per cent lower than for XCRFs, and the average labeling time for XCRFs with constraints is 50 per cent lower than for XCRFs.

4.2 Combining CRFs

Domain knowledge on the target XML application can be introduced in XCRFs by introducing constraints. XCRFs with constraints outperform XCRFs for small learning sets which is interesting in real applications. The complexity for labeling and learning is lower for XCRFs with constraints. But, the factor A occurring in the time complexity remains in the order of M^3, which may be prohibitive in the case of large sets of labels. In this section, we show how to use domain knowledge in order to combine XCRFs reducing dramatically the time complexity while preserving accuracy.

For an XML labeling task, the domain knowledge can be given by an ontology, or by DTDs or schemas over the sources in the case of XML data transformation. This allows to identify relations between labels. We propose three combination techniques for XCRFs using such relations between labels. The two first ones are basic but they allow to define the hierarchical combination for which relations between labels are defined by a hierarchy.

In the following, we assume the existence of a feature generation procedure Gen. An example of such a procedure has been given in the previous section.

4.2.1 Parallel Combination

We suppose given a partition $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_k\}$ of the set $\mathcal{Y}$ of labels. The parallel combination assumes that labelings over different parts of the partition are independent. Let us consider a special label $\perp$ not in $\mathcal{Y}$. The training algorithm 1 takes as input a sample set of pairs $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ is a tree over $\mathcal{X}$ and $\mathbf{y}$ is a tree over $\mathcal{Y}$ and a feature generation algorithm Gen. For every i, it first computes the training set S_i: for every $(\mathbf{x}, \mathbf{y})$ in S, the input tree is $\mathbf{x}$, the labeling tree is the tree $\pi_i(\mathbf{y})$ over $\mathcal{Y}_i \cup \{\perp\}$ whose symbol in position p is y_p if $y_p \in \mathcal{Y}_i$ and $\perp$ otherwise. Then, it computes the set of features F_i with the feature generation procedure Gen. Last, the training algorithm for XCRFs is applied. The output is composed of k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$ for which observed data are trees over $\mathcal{X}$ and their labelings are trees over $\mathcal{Y}_i \cup \{\perp\}$.

Algorithm 1 Parallel training

Input: a sample set S, a feature generation function Gen.

1: **for** each $i \in \{1, \ldots, k\}$ **do**
2: $S_i = \{(\mathbf{x}, \pi_i(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$ # *build* S_i *from* S
3: $F_i = \text{Gen}(S_i)$ # *build the feature set*
4: $\Lambda_i = \text{TrainXCRF}(S_i, F_i)$ # *Train an* XCRF
5: **end for**

Output: k XCRFs $\mathcal{C}_i = (F_i, \Lambda_i)$

The labeling algorithm 2 takes as input an observation $\mathbf{x}$ and k XCRFs. The k XCRFs are applied in parallel and thus independently. They produce k labelings $\mathbf{y}_i$, each of which is a tree over $\mathcal{Y}_i \cup \{\perp\}$. Last, a procedure Combine is applied to construct the labeling $\mathbf{y}$ over $\mathcal{Y}$. The procedure Combine chooses the label in $\mathcal{Y}$ with the greatest marginal conditional probability according to the k CRFs.

For parallel combination, the M^3 factor in the time complexity for inference and training is reduced to $(\frac{M}{k})^3$, assuming that subsets in the partition have roughly the same cardinalities. It should be noted that dependencies between

Algorithm 2 Parallel labeling

Input: an input $\mathbf{x}$, k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$
  for each $i \in \{1, \ldots, k\}$ **do**
2: $\mathbf{y}_i = \text{LabelXCRF}(\mathbf{x}, \mathcal{C}_i)$
  end for
Output: $\mathbf{y} = \text{Combine}(\mathbf{y}_1, \ldots, \mathbf{y}_k)$

labels which can be used by XCRFs may be lost when labels are in different subsets of the partition. Therefore, parallel combination approximates XCRFs.

4.2.2 Sequential Combination

For parallel combination, labeling problems over the $\mathcal{Y}_i$ are assumed to be independent. But it may be the case, that the labeling problems are dependent. For instance, in the label rediscovery problem for the courses dataset, if we already know that a node is labeled by `time`, its nodes should be labeled by `starttime` and `endtime`. I.e. the knowledge of labels on some nodes may help to label other nodes. Thus we present a sequential combination method where XCRFs will be used sequentially and in which labelings will be encoded in the observations to help subsequent labelings.

We suppose given an ordered list $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ of subsets of $\mathcal{Y}$ such that $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_k\}$ is a partition of $\mathcal{Y}$. Let us consider a special label $\perp$ not in $\mathcal{Y}$. The training algorithm 3 takes as input a sample set of pairs $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ is a tree over $\mathcal{X}$ and $\mathbf{y}$ is a tree over $\mathcal{Y}$ and a feature generation algorithm Gen. It first computes the XCRF $\mathcal{C}_1$ as for the parallel case. Then we iterate the construction of every $\mathcal{C}_i$ for $i \geq 2$. Let us denote by $\mathcal{Y}_{<i}$ the set $\bigcup_{1 \leq p < i} \mathcal{Y}_p \cup \{\perp\}$. Each $\mathcal{C}_i$ considers observations over the alphabet $\mathcal{X}_i = \mathcal{X} \times \mathcal{Y}_{<i}$ and labels in $\mathcal{Y}_i \cup \{\perp\}$. That is, symbols of input trees for $\mathcal{C}_i$ are pairs whose first component is the input symbol in $\mathcal{X}$ and whose second component is either a label in $\mathcal{Y}_{<i}$ or $\perp$. In order to construct $\mathcal{C}_i$, first a sample S_i is built. For an example $(\mathbf{x}, \mathbf{y})$ in S, $\pi_{<i}(\mathbf{x}, \mathbf{y})$ is the tree over $\mathcal{X}_i$ whose symbol in position p is the pair $(x_p, \pi_{<i}(y_p))$ where $\pi_{<i}(y_p) = y_p$ if $y_p \in \mathcal{Y}_{<i}$ and $\perp$ otherwise. Indeed, the problem is to find a label in $\mathcal{Y}_i$ considering that labelings for $j < i$ are known, i.e. labelings for $j < i$ are given in the input. Then, the feature generation procedure is applied. We add constraints specifying that only variables whose observation has a second component equal to $\perp$ can be labeled by a symbol in $\mathcal{Y}_i$ at step i. Last, the training algorithm for XCRFs with constraints is applied. The algorithm outputs a sequence of k XCRFs for which observed data are trees over $\mathcal{X}_i = \mathcal{X} \times \mathcal{Y}_{<i}$ and their labelings are trees over $\mathcal{Y}_i \cup \{\perp\}$.

To label an observation $\mathbf{x}$, the k CRFs are applied in sequence in algorithm 4. First $\mathbf{x}$ is labeled with $\mathcal{C}_1$. Then, for every i, we first compute $\mathbf{x}^i$ by the procedure join by introducing in $\mathbf{x}^{i-1}$ the labels obtained at step $i-1$ in $\mathbf{y}^{i-1}$. Formally, the symbol x_p^i is left unchanged if $y_p^{i-1} = \perp$, otherwise the second component of x_p^i is set to y_p^{i-1}. It should be noted that the constraints ensure that the labeling is unique avoiding the post-processing of the parallel algorithm 2. The labeling $\mathbf{y}$ is obtained as the projection over $\mathcal{Y}$ of the tree $\text{join}(x^k, y^k)$.

The complexity for inference and training is a sum over k terms (a max for parallel combination) of terms $(\frac{M}{k})^3$, assuming that subsets have the same cardinalities. Practically, this complexity is even better thanks to constraints.

Algorithm 3 Sequential training

Input: a sample set S of pairs, a feature generation function Gen.

1: $S_1 = \{(\mathbf{x}, \pi_1(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$; $F_1 = \text{Gen}(S_1)$; $\Lambda_1 = \text{TrainXCRF}(S_1, F_1)$
2: *#* $\mathcal{C}_1 = (F_1, \Lambda_1)$ *is the first* XCRF *in the output sequence*
3: **for** $i = 2$ to k **do**
4: $S_i = \{(\pi_{<i}(\mathbf{x}, \mathbf{y}), \pi_i(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$ *# build* S_i *from* S
5: $F_i = \text{Gen}(S_i)$ *# build the feature set*
6: add constraints *# only nodes with* $\perp$ *can be labeled*
7: $\Lambda_i = \text{TrainXCRF}(S_i, F_i)$ *# Train an* XCRF *with constraints*
8: **end for**

Output: A sequence of k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$

Algorithm 4 Sequential labeling

Input: an input tree $\mathbf{x}$, a sequence of k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$.

1: $\mathbf{x}^1 = \mathbf{x}$; $\mathbf{y}^1 = \text{LabelXCRF}(\mathbf{x}^1, \mathcal{C}_1)$
2: **for** $i = 2$ to k **do**
3: $\mathbf{x}^i = \text{join}(\mathbf{x}^{i-1}, \mathbf{y}^{i-1})$ *# introduce the label obtained at step* $i-1$ *in the observation*
4: $\mathbf{y}^i = \text{LabelCRF}(\mathbf{x}^i, \mathcal{C}_i)$ *# label the tree with labels in* $\mathcal{Y}_i$ *or with* $\perp$
5: **end for**

Output: $\mathbf{y} = \pi_{caly}(\text{join}(\mathbf{x}^k, \mathbf{y}^k))$ *# introduce the label obtained at step* k *and then project over* $\mathcal{Y}$ *to build the output* $\mathbf{y}$

It should be noted that, as in the parallel case, dependencies between labels are lost when labels are in different subsets of the partition. Nevertheless, in the sequential combination, labels are encoded in observations for subsequent XCRFs. This allows to consider dependencies between labels that can not be considered by a unique XCRF. Indeed, the XCRF at step i may contain features with tests over the input tree in the neighborhood. As the input tree contains labels given at previous steps, dependencies between labels in the neighborhood can be considered.

4.2.3 Hierarchical Combination

Up to now, we have introduced two combination methods for XCRFs. The parallel one assumes the independence between labels in different subsets of the partition. The sequential one allows to model dependencies between labels by introducing labels in observations for subsequent labelings. We now combine these two methods by defining the hierarchical combination. From an algorithmic point of view, algorithms in the case of hierarchical combination can easily be deduced from algorithms for parallel and sequential combination. Thus, we do not detail algorithms. Instead, we present an example to show how the hierarchical combination can come from domain knowledge and to show how the hierarchical combination generalizes over the parallel and the sequential ones.

Let us consider an XML labeling task where the labels satisfy the DTD given in Figure 8. A hierarchical presentation for the six first rules of this DTD is given in Figure 9. We use this presentation as an illustrative example in the sequel of the section.

```
<!ELEMENT movie (presentation,tagLine,plots,
                 rating,cast,technics,stories)>
<!ELEMENT presentation (fullTitle,usualTitle,year,directors,wc)>
<!ELEMENT directors (name+)>
<!ELEMENT wc (name+)>
<!ELEMENT plots (plot+)>
<!ELEMENT plot (author?,text)>
<!ELEMENT rating (p?,score,vote?)>
<!ELEMENT p (img+)>
<!ELEMENT cast (c+)>
<!ELEMENT c (actor,caracter)>
<!ELEMENT technics (runtime,country,language,
                    colors,sounds,certificates)>
<!ELEMENT colors (color+)>
<!ELEMENT sounds (sound+)>
<!ELEMENT certificates (certificate+)>
<!ELEMENT stories (trivias,goofs)>
<!ELEMENT trivias (trivia+)>
<!ELEMENT goofs (goof+)>
```

Figure 8: DTD of the Movie corpus.

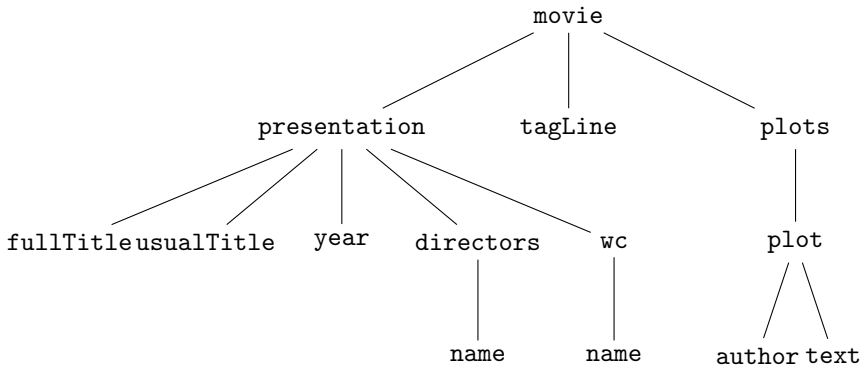

Figure 9: Hierarchical presentation of a DTD. It is a simplification of the DTD given in Figure 8 (roughly, the six first rules).

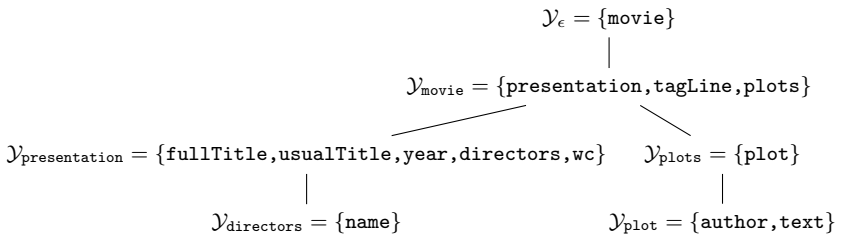

Figure 10: The hierarchy of subsets defined by the DTD (six first rules) of the Movie corpus presented in Figure 9.

The DTD can be used for defining a partition to be used in the parallel combination. For instance, one can choose to partition the set of labels as follows: one subset is defined by the root of the hierarchical presentation: $\mathcal{Y}_1 = \{\texttt{movie}\}$; other subsets are defined by labels ocuring in the subtrees under the root according to the hierarchical presentation:

- $\mathcal{Y}_2 = \{\texttt{presentation}, \texttt{fullTitle}, \ldots, \texttt{name}\}$,
- $\mathcal{Y}_3 = \{\texttt{tagLine}\}$,
- $\mathcal{Y}_4 = \{\texttt{plots}, \texttt{plot}, \texttt{author}, \texttt{text}\}$.

The DTD can be used for defining a sequence of sets of labels to be used in the sequential combination. For instance, one can choose $\mathcal{Y}_1 < \mathcal{Y}_2 < \mathcal{Y}_3 < \mathcal{Y}_4$ to order the previous partition. Or, one can choose to define subsets of labels by level and to order according to the level. On this example, we obtain the sequence: $\mathcal{Z}_1 = \{\texttt{movie}\} < \mathcal{Z}_2 = \{\texttt{presentation}, \texttt{tagLine}, \texttt{plots}\} < \mathcal{Z}_3 = \{\texttt{fullTitle}, \ldots, \texttt{plot}\} < \mathcal{Z}_4 = \{\texttt{name}, \texttt{author}, \texttt{text}\}$.

The idea behind the hierarchical combination is to combine these two methods. We consider that labels in different subtrees are independent. And we consider that there are dependencies between labels according to the child relation. Thus, we use parallel labeling for labels in different subtrees and we use sequential composition for labels at consecutive level. We define the set $\mathcal{Y}_\epsilon$ which contains the root tag and sets $\mathcal{Y}_y$ for each tag y occurring in the DTD as the set of tags occurring in the right-hand side of the rule with y as left-hand side. We can consider a hierarchical presentation of these sets. For the Movie corpus, we obtain the hierarchical presentation given in Figure 10.

The hierarchical combination is defined from such a hierarchical presentation of subsets of labels. It is defined by parallel composition at each level of the hierarchy and by sequential composition for consecutive levels in a top-down manner. On our example, $\mathcal{Y}_\epsilon$ followed by $\mathcal{Y}_{\texttt{movie}}$, followed by $\mathcal{Y}_{\texttt{presentation}}$ and $\mathcal{Y}_{\texttt{plots}}$ in parallel, followed by $\mathcal{Y}_{\texttt{directors}}$, $\mathcal{Y}_{\texttt{plot}}$ in parallel.

The hierarchical composition considers a hierarchy of sets of labels. The training algorithm and the labeling algorithm can be easily deduced from the corresponding algorithms for parallel and sequential combination. The time complexity involves a sum over all sets occurring in the hierarchy of terms m^3 where m is the cardinality of a set in the hierarchy.

#-examples	Random	DTD-based
5	83.05	**99.60**
10	84.30	**99.90**
20	79.01	**99.73**
50	79.80	**100**

Table 5: F1-measure for sequential combination of XCRFs for label rediscovery on the Movie Corpus. The column random corresponds to a randomly chosen sequence. The colum DTD-based corresponds to the sequence $\mathcal{Y}_1 < \mathcal{Y}_2 < \mathcal{Y}_3 < \mathcal{Y}_4$.

4.2.4 Empirical Evaluation

We use the Movie dataset from the "Structure Mapping Task" of the XML mining challenge[5]. It consists of 1081 XML documents of about 80 nodes each. Each document describes a movie. Each document is correct w.r.t. the DTD given in Figure 8. There are 37 tags. We consider the label rediscovery problem over this dataset.

In a first set of experiments, we compare the combination methods along the choice of the partition. For the parallel combination, the choice of the partition $\{\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3, \mathcal{Y}_4\}$ described above gives better results than a randomly chosen partition. But, the difference is not significant. For the sequential combination, we consider the sequence $\mathcal{Y}_1 < \mathcal{Y}_2 < \mathcal{Y}_3 < \mathcal{Y}_4$ described above. Experimental results are given in Table 5.

In a second set of experiments, we compare the three combination methods. Experimental results show that the sequential combination and the hierarchical one outperform the parallel one. They also show that there is no significant difference between XCRFs, sequential combination of XCRFs with the DTD-based sequence, and the hierarchical combination. We obtain similar results for the label rediscovery problem on the corpus Courses.

Moreover, the sequential combination and the hierarchical combination allow to reduce the time complexity. On the Movie label rediscovery task, the training time is divided by 30 and the labeling time is divided by 2.

Thus, the sequential combination (with a judicious choice of the sequence) and the hierarchical combination methods allow to obtain performance similar to XCRFs. And, they will allow to deal with real XML tasks in the case of large sets of labels. This is the purpose of the next section.

5 XCRFs for XML Applications

First, we consider a schema matching task. Second, we consider XML tree transformations. Last, we present a Web service R2S2 for the automatic generation of customized RSS feeds. It should be noted that XCRFs have also been used in an automatic wrapper induction system from hidden-Web sources described in Senellart et al. [2008].

[5] `http://xmlmining.lip6.fr`

5.1 Schema Matching for Real Estate I

For this problem, we evaluate XCRFs on the "Real Estate I" dataset, collected by Doan. This dataset, built from five different real estate websites, describes house listing information and contains about 10000 documents of about 35 nodes each. Each of the five sources has its own schema. A unique mediated schema with 16 tags is also known. Input trees must satisfy a source schema while target trees must satisfy the mediated schema which licences XCRFs usage.

The XML labeling task consists in labeling the nodes of the documents in their source schema with their corresponding tag in the mediated schema. For every experiment, it is supposed that three sources out of five are used for training. We ran 20 experiments, each time choosing at random 3 sources. We picked randomly 5 labeled documents from each of these 3 sources to train an XCRF. All the documents from the remaining 2 sources are the test set. Labeling one document took less than 1 second, with the XCRF having around 300 feature functions. We first evaluated the labeling of the test documents. To do so, we measured precision, recall and F1-measure for every label. We then computed the macro average over all the labels. XCRFs achieve an excellent recall of 99%, and 88% of F1-measure.

We now compare our results with the results presented by Doan et al. [2001] obtained by the LSD system. Their objective was to identify a mapping from the set of the tags of input documents schemas to the set of tags of the mediated schema. It is a simpler problem than the XML labeling task we consider. Nonetheless, we can reproduce experiments in the same conditions than in Doan et al. [2001] with XCRFs. For each of the 20 experiments we ran, we computed a mapping from the source schemas to the mediated schema using the labeled test documents. We did so by computing, for each tag in the source schemas, the tag with which it is labeled most often. This predicted mapping is then evaluated against the correct one using the accuracy measure defined in Doan et al. [2001]. The accuracy computes the proportion of matchable tags in the source schemas that are correctly matched. Using this measure, XCRFs achieve a very good 96.4% while LSD was around 80%.

5.2 XML tree transformations

We now show that XCRFs can be used to learn directly XML tree transformations.

5.2.1 XML to XML transformations

We consider the MovieDB dataset. Each document of the source describes a movie. An example is given in Figure 11. Each tree has about 160 internal nodes and 600 text nodes. The average depth is 4, the average arity is 65. The semantic of tags is rather unclear: for instance, a node labeled `CU` may contain the title. Each document of the target also describes a movie. An example is given in Figure 12. Each tree has about 41 internal nodes and 200 text nodes. We are given pairs where the first component is an XML tree of the source and the second component is its corresponding XML tree in the target. The objective is to learn an XML tree transformation between the source and the target. It should be noted that we can not use a schema matching method. Indeed, the example given in Figure 11 and 12 shows that the tag `EH` corresponds to either `director` or `actor` depending on the context in the source tree.

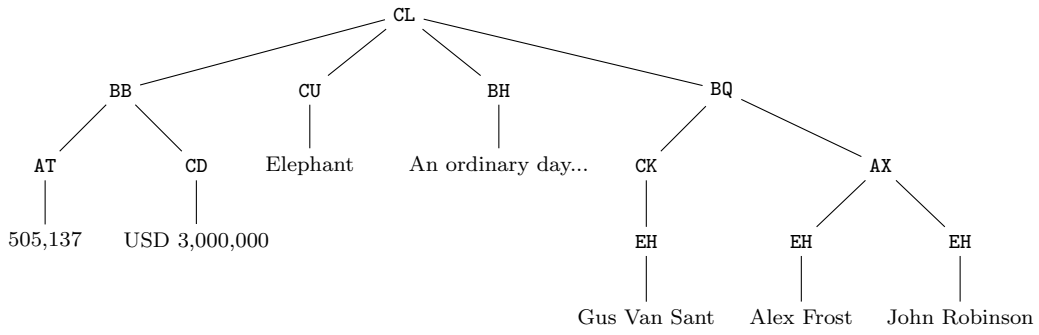

Figure 11: Part of an XML tree for the source in the MovieDB corpus.

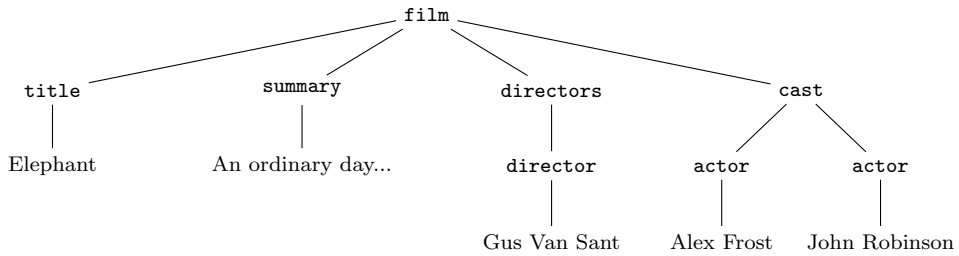

Figure 12: Part of an XML tree for the source in the MovieDB corpus.

We model an XML tree transformation by XML tree labeling of source XML trees. The target tree is embedded modulo renaming of tags in the source tree. Therefore, we consider the set $\mathcal{Y}$ of labels containing the tags of the target schema, the label `D` and the label `U`. For a source tree $\mathbf{x}$, a labeling $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{Y}$ defines uniquely a target tree $\mathbf{z}$ applying in a top-down manner the following operations: a node of $\mathbf{x}$ labeled by `D` is deleted, a node of $\mathbf{x}$ labeled by `U` is left unchanged, and a node labeled by a tag in the target schema is renamed by this tag.

Given a sample of pairs of trees $(\mathbf{x}, \mathbf{z})$ over the source and the target, a sample of pairs of trees $(\mathbf{x}, \mathbf{y})$ over the source and the corresponding labelings is constructed. There are only 22 labels in $\mathcal{Y}$ for the MovieDB corpus, thus XCRFs can be applied directly. We use 20 pairs of trees for learning and use five-fold validation. The macro-averaged F1-score is 97.65. The micro-averaged F1-score is 99.29. The difference is due to rare tags in the target. For instance, the tag label `budget` appears only in 20% of the target documents. But these experimental results only allow to evaluate the XML tree labeling task while we are trying to solve an XML transformation task.

We now evaluate results on the original XML tree transformation task. For a source tree $\mathbf{x}$, we have the correct target tree $\mathbf{z}$. The XCRF with input $\mathbf{x}$ produces a labeling $\mathbf{y}'$. We can use the labeling $\mathbf{y}'$ to define a predicted target tree $\mathbf{z}'$. The problem is now to compare $\mathbf{z}$ and $\mathbf{z}'$, for all $(\mathbf{x}, \mathbf{z})$ in the test set. For this, we compute the F1-measure according to two different criteria:

- **Paths:** for each text node in the predicted XML target tree, we build a pair made of the text content of the node and the path from the root to the node. Such a pair is compared with the pair in the correct target tree.
- **Subtrees:** for each node in the predicted XML target tree, we consider the set of all its subtrees. Such a set is compared with the set in the correct target tree.

The main difference between these two criteria is that the first one does not take into account the order in which the text leaves appear. Overall, the second criteria is more precise and gives a better idea of the accuracy of the predicted XML target trees.

The F1-score on paths is 91; the F1-score on subtrees is 88. These scores show that the XML transformation has been correctly learnt.

5.2.2 Structure Mapping Task of XML Mining Challenge

We now consider the problem of learning an XHTML to XML transformation task of larger scale. The interest is twofold. First, tags in XHTML documents have a presentation usage thus the transformation is more complex; also text nodes contain large textual values. Second, as previously we model the transformation task by a labeling task. But the set of labels is large and we must use constraints and combination techniques for XCRFs.

The task was part of the Structure Mapping task of the XML Mining Challenge[6] Denoyer and Gallinari [2006]. The dataset is made of XHTML descriptions of movies taken from the website allmovie[7] and their XML counterpart taken from the IMDB repository[8]. Each document gives thorough information about a single movie such as the title, the director, the cast and crew, related movies, *etc.*

There are two input XHTML datasets, called html1 and html2. Both datasets describe the same movies, but html1's documents contain only the XHTML table describing the movie, whereas html2's documents also contain useless information. Text nodes of an XHTML input tree may contain several text values corresponding to tags of the output. For instance, the `release_date`, the `movie_format` and the `movie_length` occur together in a text node of an input XHTML tree. Therefore, text nodes are splitted into several nodes with a tokenization algorithm. The average number of nodes in a document is 1100 in the html1 dataset, and 1250 in the html2 dataset.

Since the DTD of the output XML documents was not given as part of the challenge, a DTD was inferred with the algorithm given in Bex et al. [2006]. The inferred DTD contains 55 elements, among which 22 contain textual information. It should be noted that some values required by the output DTD may be missing in the input documents.

We model the XML tree transformation with a labeling of the source XML trees together with an algorithm constructing the target tree from a labeling and from the output DTD. For the labeling, values are contained in text nodes of the input document, therefore only leaves of source trees will be labeled. As

[6] `http://xmlmining.lip6.fr/`
[7] `http://www.allmovie.com/`
[8] `http://www.imdb.com`

leaves of the source tree have been tokenized, we consider the set $\mathcal{Y}$ of labels containing:

- the tags of the target DTD,
- the tags of type `continued` for tags in $\mathcal{Y}$ which can correspond to several leaves of source trees. For instance, the synopsis correspond to several leaves of source trees, the first leave will be labeled by `synopsis` and the following leaves by `synopsis_continued`,
- the label $\perp$ for internal nodes.

Now, given a labeling $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{Y}$, the construction algorithm defines a tree $\mathbf{z}$ by parsing $\mathbf{y}$ according to the DTD.

First, let us consider the labeling task. An XCRF can be defined. The generation procedure generates more than 7000 feature functions, there are 66 labels, the average size of source XHTML trees is 1100. Therefore, the time complexity contains the factor $7000 \times 1100 \times 66^3 \approx 2.10^{12}$ which is prohibitive. Therefore, we introduce constraints and we combine XCRFs. We introduce constraints expressing that internal nodes should be labeled by $\perp$ and only nodes containing textual values can be labeled by a label in $\mathcal{Y} \setminus \{\perp\}$. We also add constraints for labels of type `continued` because they should follow a label for the corresponding tag. We use sequential combination ordering the subsets of tags in subtrees of the target DTD.

The sequential XCRF was trained on a learning set of 692 labeled documents. We evaluated its on a testing set of 693 documents. To evaluate its performances, we measure precision, recall and F1-measure for all the 66 labels in the task. In Table 6, we only show the micro-average of these results. Since these results might be biased by the great proportion of nodes labeled with $\perp$, we also give the micro-average without this meaningless label. The micro-averaged results over all the labels are very good: both the recall and the precision are above 92%. When averaging over all the labels except $\perp$, two behaviours occur. On the one hand, the recall increases, meaning that most of the significant information in the XHTML documents are correctly labeled. On the other hand, there is a drop in precision. The explanation is that some useless information sometimes occur in the XHTML documents in a structural context very similar to that of significant information. This drop is slightly more important with the html2 dataset. This is not surprising since this dataset contains more useless information. We now evaluate results on the original XML tree transformation task.

Corpus	Average method	Rec.	Prec.	F1
html1	Micro	93.00	94.11	93.55
	Micro (without $\perp$)	94.96	77.09	85.10
html2	Micro	92.41	93.10	92.76
	Micro (without $\perp$)	94.10	69.95	80.24

Table 6: Experimental results for the XML labeling task

Second, we consider the XML transformation task. Table 7 shows the F1-scores on paths and on subtrees for the two datasets. These scores are similar

and very good for the html1 dataset. For the html2 dataset, results are a bit lower, which is consistent with the results we observed when evaluating the labeling. Still, the F1-measure on the paths of the XML documents is good and close to 80. The drop of the F1-measure on the subtrees can be explained by the nature of the html2 dataset. Indeed, with this dataset, the XCRF sometimes fails to identify useless information. Therefore, these information are in the predicted XML documents. This results in several subtrees being incorrect, and a drop of the F1-measure, although the correct information are present in the predicted XML documents.

Dataset	F1 on paths	F1 on subtrees
html1	91.81	89.76
html2	79.60	71.79

Table 7: Experimental results for the Structure Mapping Task

Thus experimental results are very promising. It should be noted that we were the only ones to work on this task and these datasets of the XML Mining Challenge. Indeed, the task is to learn a rather complex transformation. Also, the datasets require to use both the structure and the textual values of the XHTML documents. Moreover, many learning techniques can not be applied because their time complexity is too large.

5.3 R2S2: automatic generation of customized RSS feeds

The goal is to automatically generate customized RSS feeds. For instance, if a user is interested in photographs of sunsets on a photograph website, the system will allow to automatically generate an RSS feed from the search results page for the keyword "sunset". Thus we will propose a system in which the user can interactively define the wanted result and in which a program for constructing RSS feeds will be generated. First, we show how we model a program generating RSS feeds by an XML tree labeling task. Then, we describe a Web service R2S2[9] based on XCRFs for the automatic generation of customized RSS feeds.

5.3.1 Automatic Generation of RSS Feeds

The RSS feed generation from XHTML trees is a transformation of XHTML trees into XML trees. We model it by an XML tree labeling task. For this, we consider the set $\mathcal{Y}$ of labels containing: the RSS 2.0 tags, labels of type `insert` for all RSS 2.0 tags, the label `delete`, the label `delST`, and the label `U`. An example is given in Figure 3. For an XHTML tree $\mathbf{x}$, a labeling $\mathbf{y}$ of $\mathbf{x}$ in $\mathcal{Y}$ defines uniquely a target XML tree z applying in a top-down manner the following operations: a node of $\mathbf{x}$ labeled by `delete` is deleted, the subtree rooted in a node of $\mathbf{x}$ labeled by `delST` is deleted, a node of $\mathbf{x}$ labeled by `U` is left unchanged, a node labeled by a RSS 2.0 tag is renamed by this tag, and a node is inserted and labeled by the corresponding tag for a label of type `insert`. An example of output tree is given in Figure 2, it is the result of the transformation corresponding to the input tree given in Figure 1 and the labeling given in Figure 3.

[9] `http://r2s2.lille.inria.fr`

Given a sample of pairs of trees $(\mathbf{x}, \mathbf{z})$ over the source and the target, a sample of pairs of trees $(\mathbf{x}, \mathbf{y})$ over the source and the corresponding labelings is constructed. The size of $\mathcal{Y}$ is $2 \times \|\text{DTD}\| + 3$, where $\|\text{DTD}\|$ is the number of elements in the output DTD. In the experiments, we consider the RSS 2.0 specification, then the size of $\mathcal{Y}$ is 15. Thus XCRFs can be applied. Nevertheless, as the objective is to allow users to define interactively RSS feeds, the size of the sample shall be as small as possible. We have shown that constraints allow XCRFs to deal with small samples. Therefore we add constraints.

We add constraints expressing labeling properties inherent to a task of labeling for tree transformations. Indeed, to avoid incoherent labelings, it is necessary to prevent children of a node labeled with `delST` (delete the whole subtree) from being labeled with any other label than `delST`. We also add constraints given by domain knowledge. Indeed, since the output documents must satisfy the RSS 2.0 specification, several labeling dependency properties arise. For instance, the only possible children of an `item` element are `title`, `link` and `description`. Therefore, children of a node labeled with `item` or `insert_item` can not be labeled with `channel`, `item`, `insert_channel` or `insert_item`. Moreover, since the specification imposes that `title` element nodes only have a text node as a child, children of a node labeled with `title` or `insert_title` can only be labeled with `delete`, `delST` or U if the child is a text node. Adding these constraints allows to reduce the number of possible assignments from $M^3 = 15^3 = 3375$ to 831.

5.3.2 A web service R2S2

R2S2 is a Web service available at `http://r2s2.lille.inria.fr`. It allows users to define automatically customized RSS feeds. There are two steps: the *Feed Learner* and the *Feed Generator*.

The objective of the feed learner is to learn to generate RSS feeds for a Web site. An RSS feed generation program is defined by an XCRF as shown before. First, the user chooses a Web page. Then it annotates the Web page with the mandatory labels `title`, `description`, `link` and possibly the optional labels `author` and `image`. The page must be completely annotated: for instance all titles must be annotated in the Web page. To help the user, an information extraction program can generalize from partial annotations. For this the program presented in Carme et al. [2007] is implemented. Once a Web page is annotated, it can be saved and the process can be repeated. From an annotated Web page, a pair of trees $(\mathbf{x}, \mathbf{z})$, where $\mathbf{x}$ is an XHTML tree for the Web site and $\mathbf{z}$ is the corresponding XML tree, can be defined. Indeed, the knowledge of the annotation and of the RSS 2.0 specification uniquely defines an output tree $\mathbf{z}$. Therefore, a sample can be constructed and an XCRF can be learned according to the procedure described before. The learned XCRF is an RSS feeds generation program which can be saved.

The objective of the feed generator is to generate an RSS feed. For this, the user chooses an RSS feed generation program and a Web page and can generate an RSS feed adapted to the Web page. This RSS feed can then be used in his preferred feed aggregator.

6 Related Work

6.1 CRFs for Trees

As said in the introduction, CRFs have mainly be applied for sequence labeling tasks so far. The idea to define CRFs for ordered trees showed up in natural language processing only recently. Riezler et al. [2002] considered independent output variables, which corresponds to maximum entropy models (1-CRFs) for ranked sibling-ordered trees. Clark and Curran [2004], Sutton [2004] define the CRFs on derivation trees of context-free or categorial grammars. Viola and Narasimhan [2005] consider discriminative context-free grammars, trying to combine the advantages of non-generative approaches (such as CRFs) and the readability of generative ones. Very recently Finkel et al. [2008] have presented a very efficient discriminative parser based on CRFs using various optimization techniques. Cohn and Blunsom [2005] apply CRFs over the structure of each sentences syntactic parse tree to the semantic role labelling task. In the field of bioinformatics, Sato and Sakakibara [2005] propose a novel approach for RNA structural alignment based on conditional random fields. All the CRF models are used to label ordered ranked trees and dependencies between siblings have not been considered.

6.2 Constrained CRFs

Constraints have been incorporated into the Viterbi algorithm by Culota et al. [2006]. They use a constrained forward-backward algorithm for linear chain CRFs in an interactive information extraction system. In Cohn [2007], constraints have also been used for linear chain CRFs in natural language applications. More general constraints have been considered in Roth and yih [2005]. Inference is done by integer linear programming techniques, but no bounds on the running time can be given.

6.3 Web Information Extraction

Machine learning approaches to Web information extraction differ on how they model HTML documents and on the machine learning techniques used.

When dealing with token sequences Kushmerick [1997], Hsu and Dung [1998], Muslea et al. [2003], Cohen et al. [2003], information selection in HTML documents is usually reduced to a sequence labeling task. CRFs can be used in this context as in Sarawagi and Cohen [2004].

Approaches adopting the tree perspective, usually reduce information selection to XML tree labeling Carme et al. [2007], Raeymaekers et al. [2005]. Even though these algorithms for XML tree labeling are in polynomial time, they suffer from higher complexity than CRFs, since they are not linear in the size of the input trees. Also Tang et al. [2006] considers semantic annotation of Web pages as a tree labeling problem. They also consider tree structured CRFs (TCRFs). Thus their work is very similar. But they consider that the graphical structure can be a tree with cycles which is rather unclear. Therefore they use approximate algorithms for labeling and learning.

Other approaches use the layout of Web pages. Along this line of research, Zhu et al. [2005] present a two-dimensional CRF model to automatically extract information from the Web.

6.4 XML Mining

The XML mining community has put the focus on the two main tasks: XML clustering and XML classification. See for instance the INEX 2005, 2006 & 2007 challenges and the report Denoyer and Gallinari [2007]. We consider the more general XML transformation task. The main application is to transform documents in PDF or DOC in XML documents satisfying a given semantic XML schema. While it is easy to transform PDF or DOC documents into HTML or XML with non semantic tags, the difficult part is to transform XML documents with non semantic tags into XML documents with semantic tags according to a target schema.

Machine learning techniques for such XML transformation tasks were first proposed by Chidlovskii and Fuselier [2005]. They reduce the problem to tree labeling: finding semantic annotations for HTML documents according to a target XML schema. They used a two-step procedure: terminals (leaves of the output document) are predicted by a maximum entropy classifier, then the most likely output tree is generated using probabilistic parsing for probabilistic context-free grammars. Leaves in the input and the output document are supposed to be in the same order. The complexity of probabilistic parsing is cubic in the number of leaves, and therefore the system is not appropriate for large XML trees. Gallinari et al. [2005] have considered generative stochastic models for XML document transformation. Such models need to model input documents and to perform the decoding of input documents according to the learned model. Again, the complexity of the decoding procedure could be prohibitive for large XML documents.

7 Conclusion and Future Work

We adapted CRFs to XML trees by defining XCRFs. The model allows high-order features adapted to XML trees. We have presented optimization techniques adapted to XML tree labeling tasks. We have shown that XCRFs together with these optimization techniques allow to solve real-world XML applications.

In future work, we intend to extend our approach to further XML applications. For instance, XCRFs have been used in an automatic wrapper induction system from hidden-Web sources Senellart et al. [2008]. Experimental results are satisfying when HTML result pages were structured. But, it is also the case that HTML result pages pages were loosely structured. thus, it seems promising to combine linear-chain CRFs for textual contents and XCRFs for the tree structure. The problem is challenging because in this case the undirected graph would contain cycles.

Nonetheless, we consider that the problem of learning XML tree transformations remains open because we have considered simple transformations, *i.e.* XML transformations that can be defined by XML tree labeling tasks with a very restricted set of basic operations. For instance, basic operations such that switching subtrees have not been considered so far.

References

Jean Berstel and Christophe Reutenauer. Recognizable formal power series on trees. *Theoretical Computer Science*, 18:115–148, 1982.

Geert Jan Bex, Frank Neven, Thomas Schwentick, and Karl Tuyls. Inference of concise dtds from xml data. In *in Proceedings of 32nd Conference on Very Large databases - VLDB*, pages 115–126, 2006.

Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, January 2007. URL `http://hal.inria.fr/inria-00087226/en`.

Boris Chidlovskii and Jérôme Fuselier. A probabilistic learning method for xml annotation of documents. In *Proceedings IJCAI, 19th International Joint Conference on Artificial Intelligence*, 2005. URL `http://www.ijcai.org/papers/0501.pdf`.

Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 103–110, 2004.

W. Cohen, M. Hurst, and L. Jensen. *Web Document Analysis: Challenges and Opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific, 2003.

Trevor Cohn. *Scaling conditional random fields for natural language processing.* PhD thesis, University of Melbourne, 2007. URL `http://eprints.infodiv.unimelb.edu.au/archive/00002874/`.

Trevor Cohn and Philip Blunsom. Semantic role labelling with tree conditional random fields. In *CoNLL'05: Proceedings of The Ninth Conference on Natural Language Learning*, 2005.

Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available online since 1997: `http://tata.gforge.inria.fr`, October 2007. URL `http://tata.gforge.inria.fr`. Revised October, 12th 2007.

Aron Culota, Trausti Kristjansson, Andrew McCallum, and Paul Viola. Corrective feedback and persistent learning for information extraction. *Artificial Intelligence*, 170:1101–1122, 2006.

Ludovic Denoyer and Patrick Gallinari. Report on the xml mining track at inex 2005 and inex 2006. In *proceedings of INEX 2006*, 2006.

Ludovic Denoyer and Patrick Gallinari. Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. *SIGIR FORUM*, 41(1):79–90, 2007.

A. Doan. *Learning to Map between Structured Representations of Data.* PhD thesis, Univ. of Washington-Seattle, 2002. Received the ACM Doctoral Dissertation Award in 2003.

AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI magazine*, 26(1):83–94, 2005.

AnHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the ACM SIGMOD Conference*, pages 509–520, 2001.

Z. Esik and W. Kuich. Formal tree series. *Journal of Automata, Languages and Combinatorics*, 8:219 – 285, 2003.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL 2008)*, pages 959–967, 2008.

Patrick Gallinari, Guillaume Wisniewski, Ludovic Denoyer, and Francis Maes. Stochastic models for document restructuration. In *Proceedings of ECML Workshop On Relational Machine Learning*, 2005.

G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project - back and forth between theory and practice. In *23rd ACM SIGPLAN-SIGACT Symposium on Principles of Database Systems*, pages 1–12. ACM-Press, 2004.

Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM*, 51(1): 74–113, 2004. URL `http://doi.acm.org/10.1145/962446.962450`.

Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8): 521 – 538, 1998.

Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Corpus-based schema matching. In *In Workshop on Information Integration on the Web at IJCAI*, 2003.

A. McCallum and W. Li. Early results for named entity recognition with conditional random fields. In *CoNLL'2003: Proceedings of The Seventh Conference on Natural Language Learning*, 2003.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proc. 17th International Conf. on Machine Learning*, pages 591–598, 2000.

Ion Muslea, Steve Minton, and Craig Knoblock. Active learning with strong and weak views: a case study on wrapper induction. In *IJCAI 2003*, pages 415–420, 2003.

David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 235–242, 2003.

Stefan Raeymaekers, Maurice Bruynooghe, and Jan Van den Bussche. Learning (k,l)-contextual tree languages for information extraction. In *Proceedings of ECML'2005*, volume 3720 of *Lecture Notes in Artificial Intelligence*, pages 305–316, 2005.

S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 271–278, 2002.

D. Roth and W. yih. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd International Conference on Machine Learning (ICML-05)*, 2005.

Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *Proceedings of NIPS*, pages 1185–1192, 2004.

K. Sato and Y. Sakakibara. Rna secondary structural alignment with conditional random fields. *Bioinformatics*, 21(Suppl 2):237 – 242, 2005.

Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of ACM WIDM 2008 workshop on Web Information and Data Management*, 2008. URL `http://pierre.senellart.com/publications/senellart2008automatic.pdf`.

Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL*, pages 213–220, 2003.

Charles Sutton. Conditional probabilistic context-free grammars. Master's thesis, University of Massachusetts, 2004.

Charles Sutton and Andrew McCallum. *Introduction to Statistical Relational Learning*, chapter An Introduction to Conditional Random Fields for Relational Learning. MIT Press, lise getoor and ben taskar edition, 2006.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, pages 783–790, 2004.

Jie Tang, Mingcai Hong, Juanzi Li, and Bangyong Liang. Tree-structured conditional random fields for semantic annotation. In *The Semantic Web - ISWC 2006*, volume 4273, pages 640–653, 2006.

Paul Viola and Mukund Narasimhan. Learning to extract information from semistructured text using a discriminative context free grammar. In *Proceedings of the ACM SIGIR*, pages 330–337, 2005.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. Technical Report 649, UC Berkeley, Dept. of Statistics, 2003.

Hanna Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

C. S. Wetherell. Probabilistic languages: A review and some open questions. *ACM Comput. Surv.*, 12(4):361–379, 1980.

Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2d conditional random fields for web information extraction. In *Proceedings of the 22nd international conference on Machine learning*, pages 1044 – 1051, 2005.

Centre de recherche INRIA Lille – Nord Europe
Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399